Collaborative transmission in wireless sensor networks

Randomised search approaches

Stephan Sigg

Institute of Distributed and Ubiquitous Systems
Technische Universität Braunschweig

November 30, 2009

Overview and Structure

- Introduction to context aware computing
- Wireless sensor networks
- Wireless communications
- Basics of probability theory
- Evolutionary algorithms
- Cooperative transmission schemes
- Distributed adaptive beamforming
  - Feedback based approaches
  - Asymptotic bounds on the synchronisation time
  - Algorithmic improvements
  - Alternative Optimisation environments
  - A numeric approach for synchronisation
  - Consideration of node mobility

Overview and Structure

- Introduction to context aware computing
- Wireless sensor networks
- Wireless communications
- Basics of probability theory
- **Evolutionary algorithms**
- Cooperative transmission schemes
- Distributed adaptive beamforming
  - Feedback based approaches
  - Asymptotic bounds on the synchronisation time
  - Algorithmic improvements
  - Alternative Optimisation environments
  - A numeric approach for synchronisation
  - Consideration of node mobility

Outline

Randomised search approaches

1. Randomised search approaches
   - Local random search heuristics
   - Metropolis algorithms
   - Simulated annealing
   - Tabu search
2. Evolutionary algorithms
   - Restrictions of evolutionary approaches
   - Design aspects of evolutionary algorithms
3. Asymptotic bounds and approximation techniques
   - A simple upper bound
   - A simple lower bound
   - Method of the expected progress

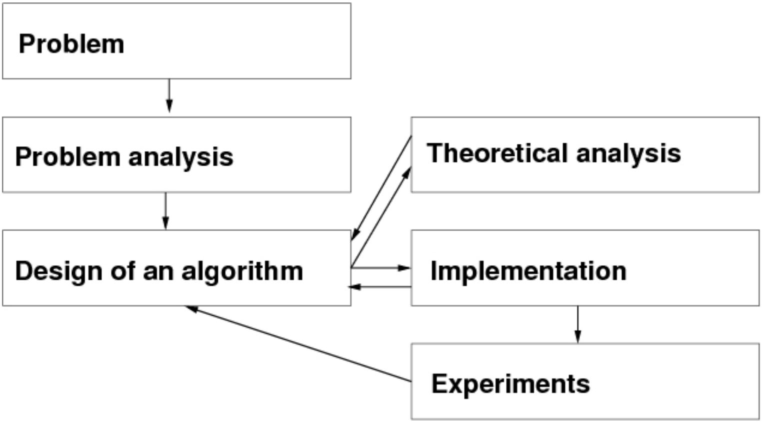
Randomised search approaches
Introduction

- Randomised search approaches
  - Combine methods that utilise random variables to guide search for optimum search point
  - Not necessarily designed for a specific problem
  - Find search point that is considered the optimum regarding a scoring function (fitness function)
  - Problem specific modelling of search space not necessarily required

Randomised search approaches

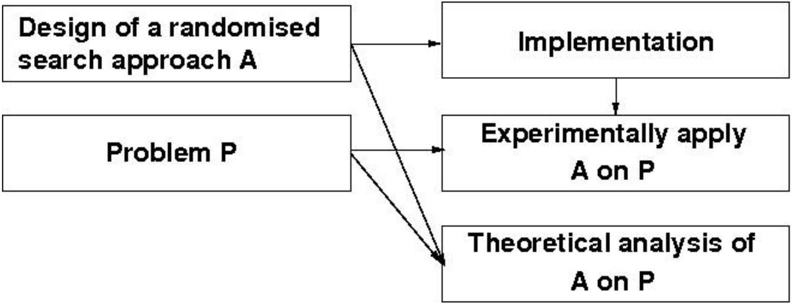
Introduction

- Classical approach to solve an optimisation problem:



Randomised search approaches
Introduction

- Random approach to solve an optimisation problem:



Randomised search approaches
Introduction

- We distinguish between
  - A search space (Genotype)
  - A feature space (Phenotype)
  - A Genotype-Phenotype-Mapping
  - A scoring function (Fitness function)
- Example
  - Genotype (binary string): 0110010
  - Phenotype (Real valued): 12

Randomised search approaches
Black-box optimisation

- Black-box optimisation:
  - Genotype-Phenotype-Mapping not known
  - Method to obtain Phenotype-outputs from Genotype-inputs (the black box) available
  - Algorithm iteratively requests Phenotype outputs for Genotype values

Randomised search approaches

Optimisation problem

- Problem formulation either maximisation or minimisation (here max):
  - Problem to solve: $max_x\{F(x)|x \in \mathbb{R}^n\}$
  - Column vector at optimum position required: $(X_1^*, x_2^*, \ldots, x_n^*)^T$
  - As well as Optimum value $F^* = F(x^*)$

Randomised search approaches

Optima

Optima

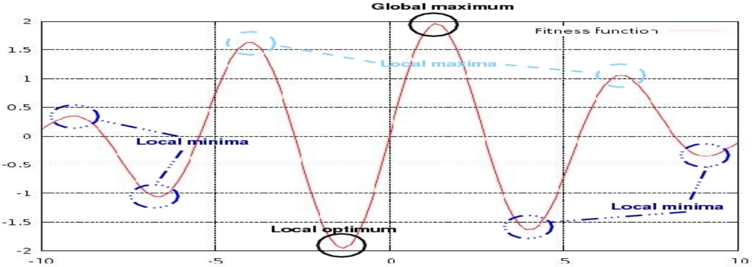
Let $f : G \rightarrow P$ be a real valued fitness function. $x^* \in G$ is an optimum point of for $\varepsilon > 0$ with $|x - x^*| < \varepsilon$ the inequality $f(x^*) \geq f(x)$ $(f(x^*) \leq f(x))$ holds.

Global optimum An optimum point x^* is called global optimum, if $f(x^*) \geq f(x)$ $(f(x^*) \leq f(x))$ for all $x \in G$.

Local optimum An optimum point which is not globally optimal is called local optimum.

Randomised search approaches

Various types of optima



- Various types of minima (maxima) can be distinguished between:
    - Local
    - Global
    - Weak
    - Strong

Randomised search approaches

Local maximum

Local maximum

For a local maximum the following conditions hold:

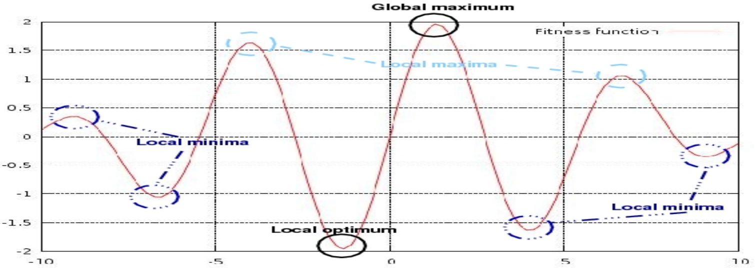
$$F(x^*) \geq F(x)$$

$$0 \leq ||x - x^*|| = \sqrt{\sum_{i=1}^{n} (x_i - x_i^*)^2} \leq \varepsilon$$

$$x \in \mathbb{R}^n$$

Randomised search approaches

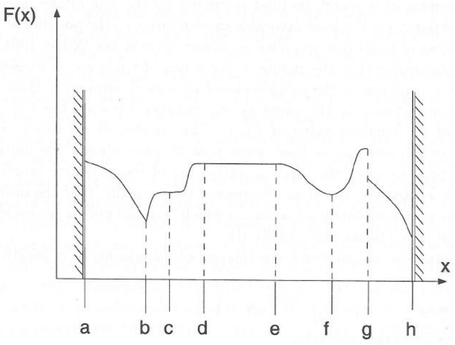
Local maximum



- The Maximum is called strong, if $F(x^*) < F(x)$ for $x \neq x^*$.
- If the objective function has only one maximum it is called unimodal
- The highest local maximum of an objective function is called the global maximum.

Randomised search approaches

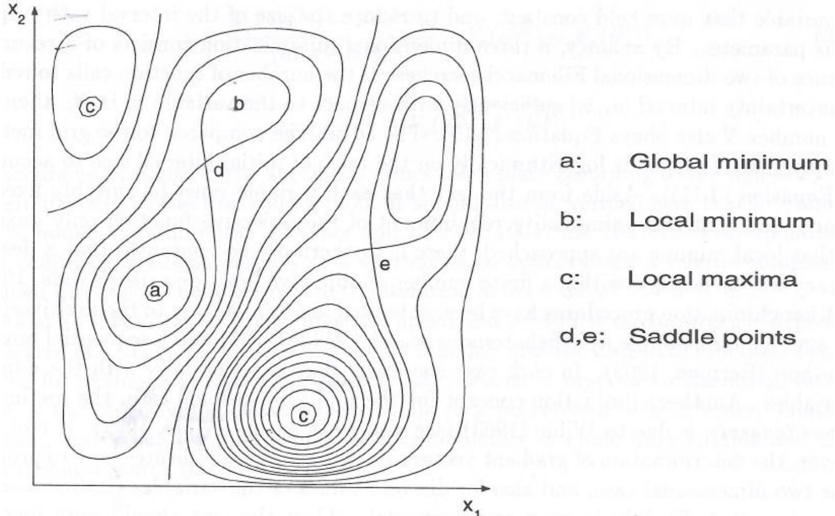
One-dimensional search problem



- Local maxima/minima: a, b, d, e, f, g, h
- Saddle point: c
- Weak local maxima: d, e
- Global maximum: g

Randomised search approaches

Multi-dimensional search problem



Randomised search approaches

Multi-dimensional search problem

- The curse of dimensionality
  - When the dimension of the search space increases linearly,
  - The number of possible solutions increases exponentially.
  - A sequential program has therefore a WC-Runtime of $O(c^n)$
    - The constant c depends on the accuracy required

Randomised search approaches

Multi-dimensional search problem

Pareto optimality

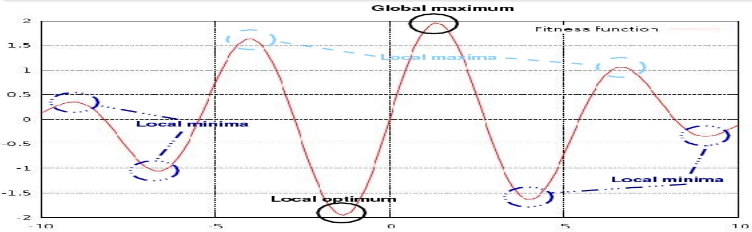
Let $\overrightarrow{x} = (x_1, \dots, x_n)^T$ be a search point in a multi-dimensional search problem and $F_i : \mathbb{R} \rightarrow \mathbb{R}, \forall i$ the objective functions for the respective dimensions. A search point $\overrightarrow{x}$ is said to be Pareto optimal with respect to a set of search points $\overrightarrow{x'} \in S$, if for at least one objective function F_i the equation $F_i(x_i) > F_i(x_i'), \forall x' \in S$ holds.

Randomised search approaches

Multimodality and unimodality

Multimodality and Unimodality

A function f is called unimodal when only one global optimum exists. Otherwise it is called multimodal.
An unimodal or multimodal function f with no local optima is called strong multimodal (unimodal). Otherwise it is called weak multimodal (unimodal).



Randomised search approaches

Local random search heuristics

- Hillclimber
- Metropolis algorithm
- Simulated annealing
- Tabu search

Local random search heuristics

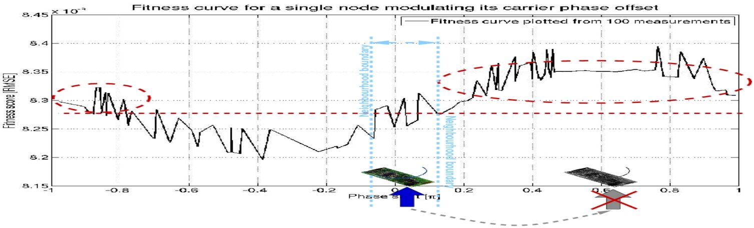
Local random search

Local random search strategies

- Intuitive way to climb a mountain (by a sightless climber)
- Most frequently applied in engineering design
  - Assumptions to state extrema are not fulfilled (e.g. unfriendly/unknown conditions)
  - Difficulties to carry out necessary differentiations
  - Solution to the equations describing all conditions does not always lead to optimum point in the search space
  - Equations to describe conditions are not immediately solvable

Local random search heuristics

Local random search

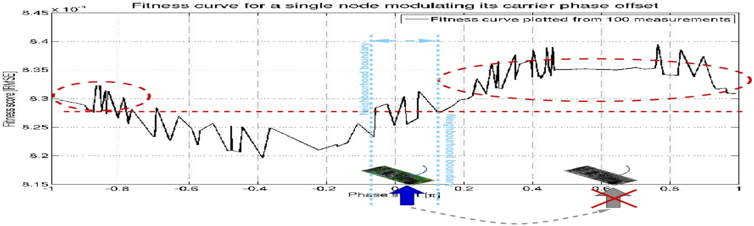


Local random search

For every point x in a search space S, a non-empty neighbourhood $N(x) \subseteq S$ is defined. The local random search approach iteratively draws one sample $x' \in N(x)$. When the fitness of the new value is better than the old one ($F(x) < F(x')$), the new value is utilised as the new best search point. Otherwise it is discarded.

Local random search heuristics

Local random search



- In principle, $N(x) = x$ or $N(x) = S$ is valid, but the original idea is that $N(x)$ is a relatively small set of search points.
- The points $x' \in N(x)$ are expected to be nearer to x than those points $x'' \notin N(x)$
- Typically, $x \in N(x)$

Local random search heuristics

Local random search

- Example: $S = \{0,1\}^n$ and $N_d(x)$ are all points y with Hamming distance smaller than d $(H(x,y) \leq d)$

$$|N_d(x)| = \begin{pmatrix} n \\ d \end{pmatrix} + \begin{pmatrix} n \\ d-1 \end{pmatrix} + \cdots + \begin{pmatrix} n \\ 1 \end{pmatrix} + \begin{pmatrix} n \\ 0 \end{pmatrix}$$

  - For constant d we obtain: $|N_d(x)| = \Theta(n^d) \ll |S| = 2^n$

Local random search heuristics

Local random search

- Local search belongs to the class of hill climbing search methods since the next search point is never chosen to decrease the fitness function.
- For deterministic random search:
  - $x' = max_{\chi}(N(x))$
  - This implies that always the highest slope is propagated

Local random search heuristics

Local random search

- Problems with local search heuristics:
  - When neighbourhood too small, easy conversion to local optima
  - When neighbourhood too big, method approximates random search
  - Therefore: Beneficial to change neighbourhood radius during optimisation
    - Initially, big neighbourhood to allow huge steps
    - Later, decrease neighbourhood size
    - Challenging: Not to decrease neighbourhood size too fast

Local random search heuristics

Local random search

- Alternative to avoid local optima: Multistart strategies
  - Local search approach applied t times on the problem domain
  - Probability amplification results in respectable search result also when single success probability is low.
    - Assume a success probability of $\delta > 0$ for one iteration of the algorithm
    - When the algorithm is applied t times, the overall probability of success is $1 - (1 - \delta)^t$
    - Small polynomial success probabilities are enough for the multistart strategy to obtain very good overall success probabilities

Local random search heuristics

Metropolis algorithms

- For the local random search heuristic, only multistart strategies are able to avoid the termination in local optima.
- A Metropolis approach allows to accept also new search points that decrease the fitness value
- If $F(x') < F(x)$ the search point x' is discarded only with probability

$$1 - \frac{1}{e^{(F(x)-F(x'))/T}}$$

Local random search heuristics

Metropolis algorithms

- Probability to accept search points with decreasing fitness value dependent on degree by which fitness decreased
- For $T \rightarrow 0$ the Metropolis approach becomes a random search
- For $T \rightarrow \infty$ the Metropolis approach becomes an uncontrolled local search
- Choice of T impacts the performance
- Knowledge on the problem or the fitness function might impact the choice of T

Local random search heuristics

Simulated annealing

- Choice of optimal T not easy: Change parameter in the pace of the optimisation
- Initially: T should allow to 'jump' to other regions of the search space with increased fitness value
- Finally: Process should gradually 'freeze' until local search approach propagates the local optimum in the neighbourhood.
- Name chosen in analogy to natural cooling processes in the creation of crystals
  - In this process, the temperature is gradually decreased so that Molecules that could move freely at the beginning are slowly put into their place

Local random search heuristics

Simulated annealing

- Optimal choice of the cooling schedule for T?
- Non-Adaptive approaches
  - Fixed temperature function $T(t)$
  - Every few steps the original value is multiplied with a factor $\alpha < 1$
- Adaptive approaches
  - React on the optimisation process
  - Probably dependent on the frequency of accepted iterations.

Random search heuristics

Simulated annealing

- Problem: No natural problem known for which it has been proved that Simulated Annealing is sufficiently more effective than the Metropolis algorithm with optimum stationary temperature.
- However, artificially constructed problems exist, for which it could be shown that Simulated Annealing is superior to the Metropolis algorithm

Random search heuristics

Tabu search

- The algorithms discussed so far only store the actual search point
- For Simulated Annealing and the Metropolis algorithm, also the search point with the best fitness value achieved so far is stored typically.
- However, knowledge about all other points is typically lost
- The algorithms might therefore access suboptimal points in the search space several times
- This increases the optimisation time

Random search heuristics

Tabu search

- Tabu-search approaches also store a list of search points that have recently been accessed.
- Due to memory restrictions the list is typically of finite length
- When the size of the list is as least of the size of the neighbourhood $N(x)$ the method can terminate when the best point in the neighbourhood has been found.

Outline

Randomised search approaches

1. Randomised search approaches
   - Local random search heuristics
   - Metropolis algorithms
   - Simulated annealing
   - Tabu search
2. Evolutionary algorithms
   - Restrictions of evolutionary approaches
   - Design aspects of evolutionary algorithms
3. Asymptotic bounds and approximation techniques
   - A simple upper bound
   - A simple lower bound
   - Method of the expected progress

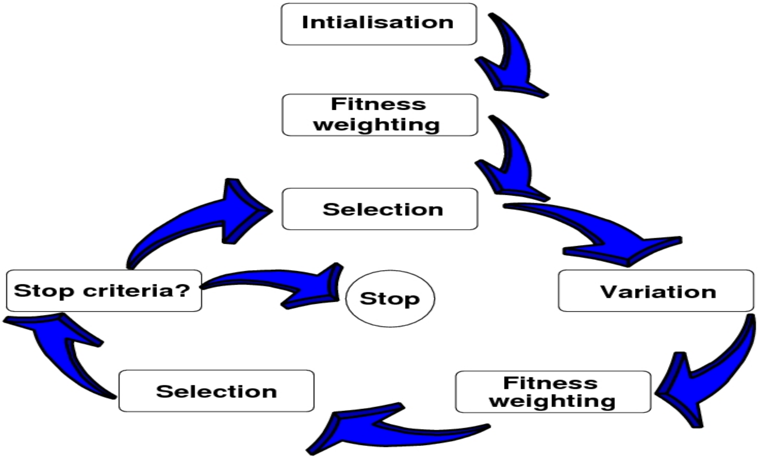
Evolutionary algorithms

Introduction

- Several researchers have studied the use of evolutionary approaches for optimisation purposes
- To-date, evolutionary algorithms combine these different approaches so that no clear distinction can be made
- An overview on various approaches is given in the following

Evolutionary algorithms

Introduction



Evolutionary algorithms

Genetic algorithms

- Proposed by John Holland [1]
- Binary discrete search spaces: $\{0,1\}^n$
- Fitnessproportional selection
  - For m individuals $x_1, \ldots, x_m$ the probability to choose x_i is $\frac{f(x_i)}{f(x_1)+\cdots+f(x_m)}$.
- Main evolution operator is crossover
  - Originally One-point crossover
- The main goal was not optimisation but the adaptation of an environment

[1] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

Evolutionary algorithms

Genetic algorithms

- The hope associated with genetic algorithms was that they are able to solve some functions especially well

Separable function

A function is called separable, if the input variables can be divided into disjoint sets $X_1, \ldots, X_k$ with $f(x) = f_1(X_1) + \cdots + f_k(X_k)$

- Since genetic algorithms utilise crossover, it was expected that they are therefore well suited to quickly find the optimum on separable functions

Evolutionary algorithms

Genetic algorithms

Royal road functions

k blocks of variables of length l are formed. On each block X_l the identical function f_l is implemented with

$$f_l(X_l) = \begin{cases} 1 & \text{All variables in } X_l \text{ equal } 1 \\ 0 & \textit{else}. \end{cases} \tag{1}$$

- It was shown that genetic algorithms do NOT perform well on these functions.[2]
- The reason is that it is highly unlikely to perform crossover exactly at the border of the variable blocks.
- It is better to optimise the single blocks on their own separately by mutation.

[2] T. Jansen and I. Wegener, *Real royal road functions – where crossover provably is essential*, Discrete applied mathematics, Vol. 149, Issue 1-3, 2005.

Evolutionary algorithms

Evolution strategies

- Proposed by Bienert, Rechenberg and Schwefel[3] [4]
- At first only steady search spaces as $\mathbb{R}^n$
- No Crossover
- Only mutation
  - First mutation operator: Each component x_i is replaced by $x_i + \sigma Z_i$ (Z_i normally distributed, σ^2 Variance)

---

[3] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, 1973.*

[4] H.P. Schwefel, *Evolution and optimum seeking*, 1993

Evolutionary algorithms

Evolution strategies

1/5 rule

After $10n$ iterations, the variance is adopted every n iterations. When the number of accepted mutations in the last $10n$ steps is greater than $1/5$, σ is divided by 0.85 and else multiplied by 0.85.

- This heuristic is based on an analysis of the fitness function $x_1^2, \dots, x_n^2$ – the sphere model.

Evolutionary algorithms

Evolutionary programming

- The approach was proposed by Lawrence J. Fogel[5][6]
- Various similarities to evolution strategies
- Search Space: Space of deterministic finite automata that well adapt to their environment.

[5] L.J. Fogel, *Autonomous automata*, Industrial Research, Vol. 4, 1962.

[6] L.J. Fogel *Biotechnology: Concepts and Applications*, Prentice-Hall, 1963

Evolutionary algorithms

Genetic programming

- Proposed by John Koza[7]
- Search space: Syntactically correct programs
- Crossover more important than mutation

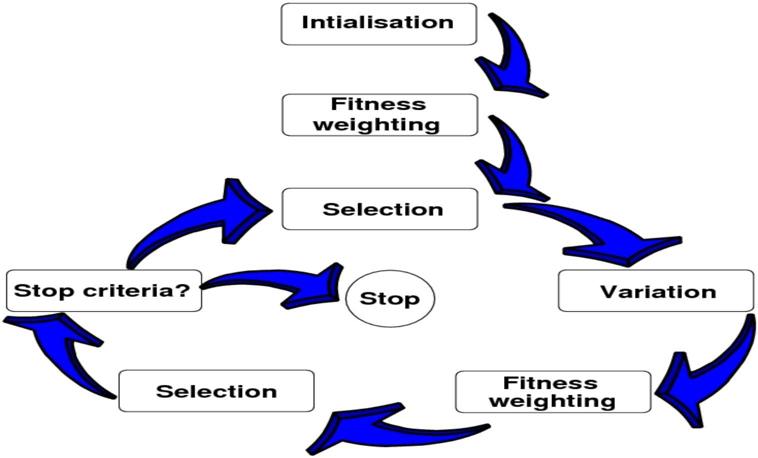
[7] John Koza *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992

Evolutionary algorithms

Hybrid approaches

- Since evolutionary approaches are typically slow to initially find a search point with a reasonable fitness value,
- Approaches are combined with fast heuristics that initially search for a good starting point.
- Afterwards the evolutionary approach is applied

Evolutionary algorithms
Modules

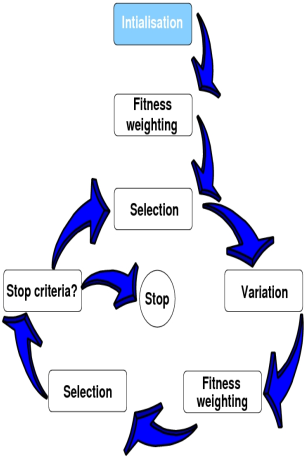


Evolutionary algorithms

Modules

Initialisation

- Initialise μ individuals from the search space S
- Typically uniformly at random
- Typical search spaces: $S = \mathbb{R}^n$ or $S = \mathbb{B}^n$
- Achieve sufficient coverage:
  - Distance measure d
  - distance $\geq d$
- Improve optimisation time and quality of solution:
  - fast heuristics for individual population

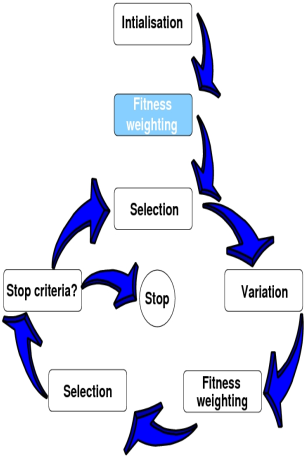


Evolutionary algorithms

Modules

Fitness weighting of the population

- Individuals of population weighted for their fitness value.
- Fitness function $f : S \to \mathbb{R}$
- Monotonous function

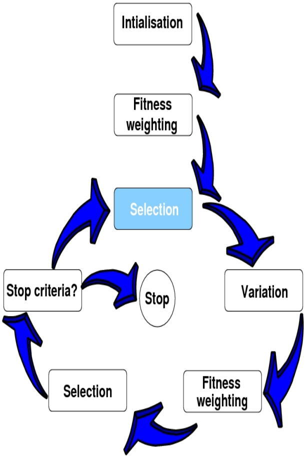


Evolutionary algorithms

Modules

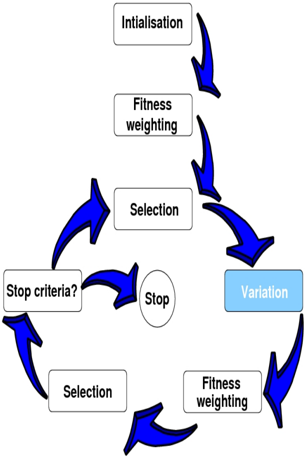
Selection for reproduction

- Dependent on fitness values reached by individuals
- individuals chosen to produce offspring population
- Intuition:
  - Individuals with good fitness value: Higher probability to produce high-rated individuals for offspring population



Evolutionary algorithms

Modules



Variation

- Offspring population created by mutation and/or crossover.
- Mutation is typically local search operator
- Crossover allows to find search points in currently not populated regions
- Adaptive implementations possible

Evolutionary algorithms

Modules

Mutation

- Produces individuals that differ only slightly from the parent-individuals.
- One parent individual produces one offspring individual
- Mutation operators differ between search spaces.

Evolutionary algorithms

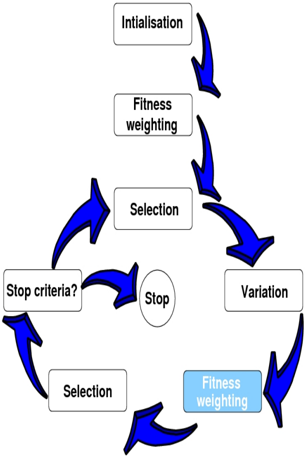
Modules

Crossover

Crossover is a variation technique that produces one or more offspring individuals from two or more parent individuals

Evolutionary algorithms
Modules

- All newly generated offspring individuals are weighted by a fitness function f.
- Structure of f impacts performance of random search approach
  - Weak multimodal vs. strong multimodal

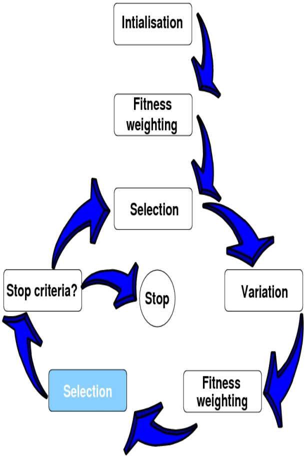


Evolutionary algorithms

Modules

Selection for substitution

- Population size increased due to variation
- Reduce population size to μ
- Typically: Individuals with higher fitness values more probable



Evolutionary algorithms

Modules

$+$ and , strageties

$(\mu + \lambda)$ strategies: Offspring population chosen from μ old individuals '$+$' λ offspring individuals

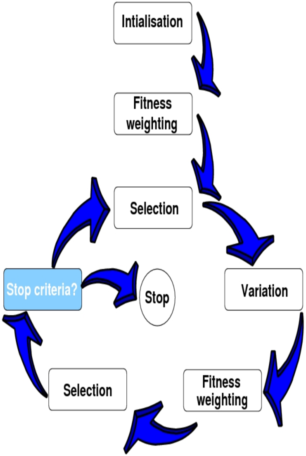
(μ, λ) strategies: μ individuals drawn from λ offspring individuals while μ old individuals are discarded

- Comma-strageties try to avoid local optima

Evolutionary algorithms

Modules

- Since global optimum not known, stop criteria required



Outline

Randomised search approaches

1. Randomised search approaches
   - Local random search heuristics
   - Metropolis algorithms
   - Simulated annealing
   - Tabu search
2. Evolutionary algorithms
   - Restrictions of evolutionary approaches
   - Design aspects of evolutionary algorithms
3. Asymptotic bounds and approximation techniques
   - A simple upper bound
   - A simple lower bound
   - Method of the expected progress

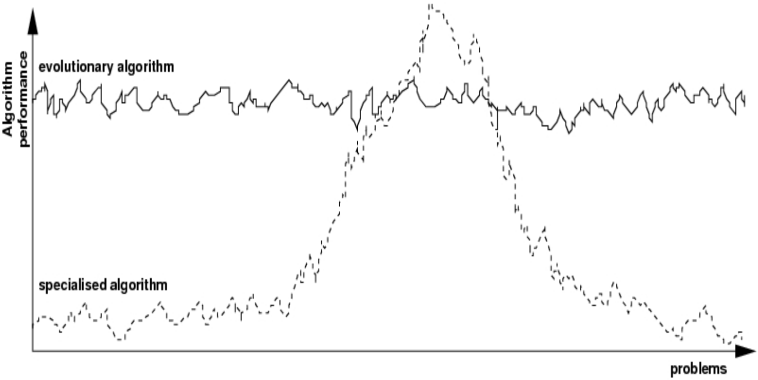
Restrictions of evolutionary approaches

The No-free-lunch theorem

- In the early days of evolutionary algorithm it has been argued that
  - Problem specific algorithms are better than evolutionary algorithms on a very small subset of problems
  - Evolutionary algorithms perform better on average over all problems
- Therefore, evolutionary algorithms have been proposed as a good choice for a general purpose optimisation scheme

Restrictions of evolutionary approaches

The No-free-lunch theorem



Restrictions of evolutionary approaches

The No-free-lunch theorem

- Can an algorithm be suited for 'all' problems?
  - Distinct coding of the search space
  - Various fitness functions
- What does 'all problems' mean?
  - For all possible representations and sizes of the search space
  - All possible fitness functions on the feature space
  - For a given search space and feature space, all possible fitness functions
  - Every single point in the search space is the optimum point in several of these problems
- Can one algorithm be better on average than another algorithm on 'all' problems?

Restrictions of evolutionary approaches

The No-free-lunch theorem

- To understand this scenario, Wolpert and Macready formalised the assertion[8]
- Assumptions:
  - The set of all functions $f : S \rightarrow W$ considered is given by F
  - S and W are finite (as every computation on physical computers can only have finite resources)
  - The fitness function is evaluated only once for each search point
  - $A(f)$ is the number of search points requested until the optimum is found

[8] D.H. Wolpert and W.G. Macready, *No Free Lunch Theorems for Optimisation*, IEEE Transactions on Evolutionary Computation 1, 67, 1997.

Restrictions of evolutionary approaches

The No-free-lunch theorem

No free lunch theorem

Assume that the average performance of an algorithm in the No Free Lunch Scenario for S and W is given by $A_{S,W}$, the average over all $A(f), f \in F$. Given two algorithms A and A', we obtain $A_{S,W} = A'_{S,W}$

- This means that two arbitrary algorithms perform equally well on average on all problems

Restrictions of evolutionary approaches

The No-free-lunch theorem

Proof of the No Free Lunch Theorem

Proof by induction over $s := |S|$.
W.l.o.g.: $W = \{1, \ldots, N\}$
We consider sets $F_{s,i,N}$ of all functions f on a search space of non-visited search points of size s with at least one x with $f(x) > i$
Observe that for every function f and every permutation π also f_π belongs to $F_{s,i,N}$

Restrictions of evolutionary approaches

The No-free-lunch theorem

Proof of the No Free Lunch Theorem

Induction start: $s = 1$

Every algorithm has to choose the single optimum search point with its first request.

Restrictions of evolutionary approaches

The No-free-lunch theorem

Proof of the No Free Lunch Theorem

Induction: $s - 1 \longrightarrow s$

We define a function $a : S \longrightarrow \mathbb{N}$ so that for every $x \in S$ the share of functions with $f(x) = j$ is exactly $a(j)$.

This is independent of x, since all permutations f_π of a function f also belong to $F_{s,i,N}$,

$a(j)$ is therefore the probability to choose a search point with fitness value j – Independent of the concrete algorithm A

Restrictions of evolutionary approaches

The No-free-lunch theorem

Proof of the No Free Lunch Theorem

Induction: $s - 1 \rightarrow s$

With probability $a(j)$ an algorithm A finds a search point with fitness value j.

If $j > i$, the number of functions $f(x) = j$ is equal to the number of functions $f_\pi(y) = j$, since all permutations of f are also in $F_{s,i,N}$.

The probability to achieve a fitness value $j > i$ is therefore independent of the algorithm.

Restrictions of evolutionary approaches

The No-free-lunch theorem

Proof of the No Free Lunch Theorem

Induction: $s-1 \rightarrow s$

With probability $a(j)$ an algorithm A finds a search point with fitness value j.

If $j \leq i$, x is not optimal in scenario $F_{s,i,N}$ and the new scenario is $F_{s-1,i,N}$

Restrictions of evolutionary approaches

The No-free-lunch theorem

Proof of the No Free Lunch Theorem

Summary – in other words:
For any two algorithms we can state a suitable permutation of the Problem-function for one problem (i.e. state another problem), so that both algorithms in each iteration request identical search points.

- Especially, since every search point could be optimal, there are always algorithms that request the optimal search point right from the start.

Restrictions of evolutionary approaches

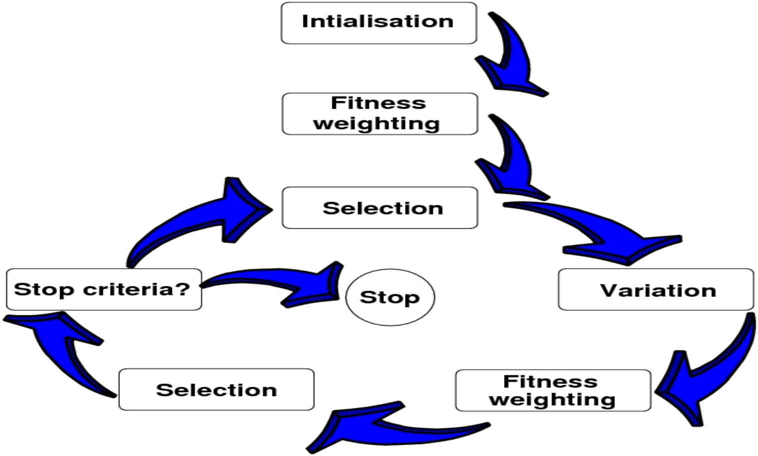
An almost-no-free-lunch-theorem

- The NFL is possible, since ALL algorithms and ALL problems are considered
- It is a reasonable question if an NFL is also valid in smaller, more realistic scenarios.
- In [9] is was proved, that a similar theorem can be stated also for more realistic problem scenarios.

[9] S. Droste, T. Jansen and I. Wegener, *Perhaps not a free lunch but at least a free appetizer*, Proceedings of the 1st Genetic and Evolutionary Computation Conference, 1999.

Design aspects of evolutionary algorithms

Overview



Design aspects of evolutionary algorithms

Search space

- Design of search space has great impact on the performance of an algorithm
- Which parameters impact the fitness by what amount
- Parameters might depend on each other so that not all have to be modelled

Design aspects of evolutionary algorithms

Search space

- Often natural to represent search points as vectors
  - Components of the same set $(\mathbb{R}, \mathbb{Z}, \mathbb{N}, \{0, 1\})$
  - Leads to search spaces of the type $S = X^n$
  - Also vectors with components of distinct type possible (multi-type)
- Mutation and crossover operators have to respect these properties of the search space.
- Mutation and crossover often assume that neighbouring search points are related to each other.
- Important to choose a representation that well reflects the characteristics of the problem at hand.

Design aspects of evolutionary algorithms

Search space

Hamming cliff

- The hamming distance between 2^n and $2^n + 1$ is 1
- The hamming distance between 2^n and $2^n - 1$ is $n + 1$!!!

- A possible solution are Gray Codes
- The hamming distance between neighbouring numbers is always one

Design aspects of evolutionary algorithms

Search space

Gray codes

- For the numbers 0 and 1, the representation is 0 and 1
- When $0, \dots, 2^n - 1$ are correctly represented by the bitvectors $a_0, \dots a_{N-1}$ with $N = 2^n$
  - Represent $0, \dots, 2^{n+1} - 1$ by $0a_0, \dots, 0a_{N-1}, 1a_{N-1}, \dots, 1a_0$
- The hamming distance of neighbouring numbers is then 1

- The drawback of this approach is that numbers with greater numerical distance have also to distance 1
- $0a_0$ and $1a_0$ also have hamming distance 1

Design aspects of evolutionary algorithms

Selection principles

- Selection principles rule which individuals are the basis for the next generation.
- The selection is based on the fitness function
- Often: Survival of the fittest

Design aspects of evolutionary algorithms

Selection principles

- Selection strategies
  - Try to optimise the overall fitness of individuals
    - Assume: Individuals with similar fitness values are neighbours in the search space
  - Try to prevail diversity in the search space
- Both strategies are contradictory

Design aspects of evolutionary algorithms

Selection principles

Uniform selection

Individuals chosen uniformly at random

Deterministic selection

Deterministically choose the highest rated individuals for the selection

Threshold selection

Candidates for offspring population drawn uniformly at random from the t highest rated individuals

Design aspects of evolutionary algorithms

Selection principles

Fitnessproportional selection

- For population $x_i, \ldots, x_n$ individual x_i chosen with

$$p(x_i) = \frac{f(x_i)}{f(x_1) + \cdots + f(x_n)}$$

- Draw random variable u from $[0, 1]$ and consider x_i if

$$p(x_1) + \cdots + p(x_{i-1}) < u \leq p(x_1) + \cdots + p(x_i)$$

- Frequently applied for evolutionary approaches

Design aspects of evolutionary algorithms

Selection principles

- Problems with Fitnessproportional selection
  - Linear modification of the fitness function ($f \rightarrow f + c$) results in different behaviour
  - When fitness values sufficiently separated, selection is nearly deterministic
  - When deviation in fitness values is small relative to absolute values, similar to uniform selection

Design aspects of evolutionary algorithms
Selection principles

Tournament selection

- A tournament size of $q \in \{1, \ldots, n\}$ is defined.
- A set of q individuals is then drawn uniformly at random from the population
- The best individual from this set is considered for the offspring population.

- For $q = 1$ the tournament selection is a random selection
- For $q = n$ it implements a deterministic choice
- Also individuals with non-optimal fitness values are consideredchosen

Design aspects of evolutionary algorithms

Selection principles

SUS – Stochastic Universal Sampling

- All individuals are enumerated
- The λ individuals are drawn equally distributed from this enumeration
- The first individual is determined randomly from $[0, 1/\lambda)$
- The further $\lambda - 1$ individuals follow at distance $\frac{1}{\lambda}$

- SUS is a selection mechanism especially proposed for evolutionary algorithms
- λ candidates for the offspring population are created in the manner described

Design aspects of evolutionary algorithms

Selection principles

- Some selection approaches have problems with the scaling of the fitness function (e.g. fitness proportional selection)
- The $(\mu + \lambda)$ and (μ, λ) strategies fall into this category.
- Also: Threshold selection

Design aspects of evolutionary algorithms

Selection principles

- Lifetime of individuals
  - Some strategies define a maximum lifetime of individuals
  - An individual is then replaced when its maximum lifetime is reached
- Most approaches implement unlimited lifetime
- For comma strategies the lifetime is 1 for every individual

Design aspects of evolutionary algorithms

Selection principles

- Since a great number of distinct selection strategies exists, a quality measure for selection strategies is desired.

Design aspects of evolutionary algorithms

Selection principles

Quality measure – Takeover time

The takeover time is the count of generations until an algorithm that exclusively relies on selection (no mutation or crossover) has replaced all individuals in the population by the best individual

- Very short or very long takeover times are not good
- Algorithm then either not converges or converges in local optima
- But even when the takeover time is known it is still not clear how to interpret the data

Design aspects of evolutionary algorithms

Selection principles

Quality measure – Selection intensity

To calculate selection intensity, the variance σ^2 of the fitness values in the population and the mean fitness value is measured before $(\overline{f})$ and after $(\overline{f_{sel}})$ the selection.
The selection intensity is then defined as

$$I = \frac{\overline{(f_{sel} - \overline{f})}}{\sigma}$$

- Measure depends on the variance of the fitness values
- Variance of fitness values dependent on selection method
  - Quality measure therefore depends on selection method that is to be quantified.
- Interpretation of this measure is therefore not trivial

Design aspects of evolutionary algorithms
Mutation

- A mutation creates one offspring individual from one given individual
- Mutation operators are designed for specific search spaces
- Mutation shall apply only few modifications of individuals on average
- Individuals that are closer to the original individual (regarding the neighbourhood function) shall have a greater probability than those that are farther away

Design aspects of evolutionary algorithms

Mutation

- Search spaces in $\{0,1\}^n$
  - Common mutation operator chooses mutation probability p for each bit
  - To obtain a search point with hamming distance i the probability is $p^i(1-p)^{n-i}$
  - $p = \frac{1}{2}$ is random search
  - To assure that individuals that are farther away have decreased probability to be constructed, $p \leq \frac{1}{2}$
  - The expectation on the number of bits mutated is np and the variance is $np(1-p)$
  - Unlikely to obtain individual far away in the search space
  - A standard choice is $p = \frac{1}{n}$

Evolutionary algorithms

Modules

Mutation operators for individuals from $\mathbb{B}^n$:

Standard bit mutation

- Offspring individual created bit-wise from parent individual
- Every bit 'flipped' with probability p_m
- Common choice: $p_m = \frac{1}{n}$

1 bit mutation

- Offspring individual identical in all but one bit.
- This bit chosen uniformly at random from all n bits

Design aspects of evolutionary algorithms

Mutation

- Search spaces $A_1 \times \cdots \times A_n$
  - A similar approach as for $\{0, 1\}$ search spaces can be taken
  - With probability p one of $|A_i|$ possible values is taken uniformly at random for position i
  - The probability that position i is not mutated is therefore

$$(1-p) + p \cdot \frac{1}{|A_i|}$$

Design aspects of evolutionary algorithms
Mutation

- Search space $\mathbb{R}^n$
  - For mutation purposes, a probability vector is typically added to the actual search point
  - The expectation of the vector should be 0 so that no direction is preferred

Evolutionary algorithms

Modules

Mutation operators for $\mathbb{R}^n$:

- Offspring individual generated by adding a vector $m \in \mathbb{R}^n$ to parent individual

  Restricted mutation :
  Vector in restricted interval: $v_i \in [-a, a]$

  Unrestricted mutation :
  $v_i \in \mathbb{R}$

Design aspects of evolutionary algorithms

Mutation

- Permutations on the search space
  - Example: TSP – k-opting
    - Order of places is unravelled at k positions
    - These k blocks are then again connected randomly
  - Another approach is to change the order of nodes in some blocks

Design aspects of evolutionary algorithms
Mutation

- Mutations of syntax trees (Genetic programming)
  - One of four possible mutation operators is chosen uniformly at random
    - **Grow** Choose a leaf and replace this by random syntax tree
    - **Shrink** Choose an inner node and replace this by a leaf with random value
    - **Switch** Choose random inner node and exchange the position of two randomly chosen children
    - **Cycle** Choose a node at random and change its labelling/value
  - It has to be taken care that the resulting syntax tree remains syntactically correct

Design aspects of evolutionary algorithms
Recombination

- Recombination typically takes two individuals and results in one or two offspring individuals
  - Also recombination of more than two individuals possible
  - Often generalisations of the two-individual case
- Distinct recombination methods for various search spaces
- Crossover parameter p_c specifies the probability with which crossover (and not mutation) is applied for one selected individual
- In some cases (e.g. binary coded numbers) not all positions in the individual string are allowed to apply crossover on

Design aspects of evolutionary algorithms

Recombination in $\{0,1\}^n$

- One-point crossover
- k-point crossover
- Uniform crossover

Evolutionary algorithms
Modules

Crossover operators for $\mathbb{B}^n$:

One-point crossover: Individual x'' from two individuals x and x' according to uniformly determined crossover position:

$$x''_j = \begin{cases} x_j & \text{if } j \leq i \\ x'_j & \text{if } j > i \end{cases} \tag{2}$$

Evolutionary algorithms

Modules

Crossover operators for $\mathbb{B}^n$:

k-point crossover: Choose $k \leq n$ positions uniformly at random:

$$
\begin{array}{rl}
x_1 = & x_{11}, x_{1,2}, \dots, x_{1,k_1} | x_{1k_1+1}, \dots, x_{1k_2} | x_{1k_2+1}, \dots, x_{1n} \\
x_2 = & x_{21}, x_{2,2}, \dots, x_{2,k_1} | x_{2k_1+1}, \dots, x_{2k_2} | x_{2k_2+1}, \dots, x_{2n} \\
\hline
y_1 = & x_{11}, x_{1,2}, \dots, x_{1,k_1} | x_{2k_1+1}, \dots, x_{2k_2} | x_{1k_2+1}, \dots, x_{1n} \\
y_2 = & x_{21}, x_{2,2}, \dots, x_{2,k_1} | x_{1k_1+1}, \dots, x_{1k_2} | x_{2k_2+1}, \dots, x_{2n}
\end{array}
$$

Evolutionary algorithms

Modules

Crossover operators for $\mathbb{B}^n$:

Uniform crossover: Each bit chosen with uniform probability from one of the parent individuals

Design aspects of evolutionary algorithms

Recombination in $\{0,1\}^n$

Shuffle crossover

- Parent-individuals are randomly permutated with π
- Crossover operation is applied
- Resulting individuals are re-permutated with π^{-1}

- For shuffle crossover, neighbouring bits have not a higher probability to have their origin in the same parent individual

Design aspects of evolutionary algorithms

Recombination in $\{0,1\}^n$

Random respectful recombination

- All information that is identical in both parent individuals is copied to the child-individual
- For all other positions, the value is chosen uniformly at random

Evolutionary algorithms
Modules

Crossover operators for $\mathbb{R}^n$:

1-point crossover: Analogous to 1-point crossover in $\mathbb{B}^n$

k-point crossover: Analogous to k-point crossover in $\mathbb{B}^n$

Uniform crossover: Analogous to uniform crossover in $\mathbb{B}^n$

Arithmetic crossover: Individual $\mathcal{I} \in \mathbb{R}^n$ weighted sum from k parents $x_1, \ldots, x_k$:

$$\mathcal{I} = \sum_{i=1}^{k} \alpha_i x_i; \text{ with } \sum_{i=1}^{k} \alpha_i = 1$$

Design aspects of evolutionary algorithms

Recombination in $\mathbb{R}^n$

Alternative recombination approaches in $\mathbb{R}^n$

- When parent individuals have values x_i and y_i at position i
- We can choose position i for the child as

$$x_i + \mu_i(y_i - x_i) \tag{3}$$

- μ_i is drawn uniformly at random from $[0, 1]$

Design aspects of evolutionary algorithms

Recombination for permutations

Order crossover

- Variant of two-point crossover that is suitable for permutations
- Values between both crossover positions are taken from the first individual
- All missing values are filled in the order they occurred in the second individual (beginning from the second crossover position)

Parent 1	12	3456	789
Parent 2	84	1593	627
Child	??	3456	???
Child	19	3456	278

Design aspects of evolutionary algorithms

Recombination for permutations

Partially mapped crossover (PMX)

- Variant of two-point crossover that is suitable for permutations
- Values between both crossover positions are taken from the first individual
- Missing values are included at the same position the value is found in the second individual.
- If this position is already occupied by value x_i, the position of individual x_i is chosen instead (and so on)

Parent 1	12	3456	789
Parent 2	84	1593	627
Child	??	3456	???

Child	89	3456	127

Design aspects of evolutionary algorithms

Recombination for permutations

Order crossover II

- k positions are randomly marked
- All other positions are taken over from the second parent in their occurrence order

- Assume that the positions 2,4,6,8 are marked.

Parent 1	12	3456	789
Parent 2	$8\underline{4}$	$1\underline{5}9\underline{3}$	$6\underline{2}7$
Child	82	1394	657

Design aspects of evolutionary algorithms

Structures of populations

- The structure of the population has also an impact on the performance of the algorithm
  - Consideration of duplicate individuals
  - Diversity

Design aspects of evolutionary algorithms

Structures of populations

- Creation of niche in the population
  - In order to keep isolated individuals with respectable fitness value
  - The number of individuals in the neighbourhood is also considered for the fitness-based selection

$$f'(x) = \frac{f(x)}{d(x, P)} \tag{4}$$

Design aspects of evolutionary algorithms

Structures of populations

- Consideration of sub-populations
  - Similar individuals are grouped together for optimisation
  - Recombination not over the whole population but between individuals of a sub population
  - Idea:
    - Individuals of distinct sub-populations have good fitness.
    - By crossover operation, an individual in between is created that has typically worse fitness value
  - Selection applied on the overall population

Design aspects of evolutionary algorithms

Dynamic and adaptive approaches

- As parameter choices impact the performance of an evolutionary algorithm, adaptation of parameters during simulation might also be beneficial
- Similar approach as for the 'mutation' probability of simulated annealing
- Feasible also for Crossover, mutation, fitness function, population structure

Design aspects of evolutionary algorithms

Comments on the implementation of evolutionary algorithms

- Evolutionary algorithms are easy to implement when compared to some complex specialised approaches
- However, Evolutionary algorithms are computationally complex
- It is therefore beneficial to implement efficient variants to the distinct methods

Design aspects of evolutionary algorithms

comments on the implementation of evolutionary algorithms

- Generation of pseudo random bits is important for many of the theoretic results for evolutionary algorithms to hold
- It is, however possible to reduce the number of random experiments
  - It is more efficient to calculate the next flipping bit in a mutation instead of doing the calculation for every bit independently

Design aspects of evolutionary algorithms

comments on the implementation of evolutionary algorithms

- Most of the computational time is typically consumed by the fitness calculation
- One approach to reduce complexity is to prevent re-calculation of fitness for individuals
  - Dynamic data structures that support search and insert

Asymptotic bounds and approximation techniques

A simple upper bound

Method of the fitness based partition

- Simple method to provide an upper bound on the expected optimisation time
- Applicable to random search schemes with 'plus' selection
- Exemplarily for the $(1+1)$-EA

Asymptotic bounds and approximation techniques

A simple upper bound

Fitness-based partition

Let $f : \mathbb{B}^n \rightarrow \mathbb{R}$ be a fitness function. A partition $L_0, L_1, \ldots, L_k \subseteq \mathbb{B}^n$ with $\mathbb{B}^n = L_0 \cup L_1 \cup \cdots \cup L_k$ is a fitness based partition of f when

1. $\forall i, j \in \{0, \ldots, k\}, \forall x \in L_i, y \in L_j : (i < j \Rightarrow f(x) < f(y))$ and
2. $L_k = \{x \in \mathbb{B}^n | f(x) = \max \{f(y) | y \in \mathbb{B}^n\}\}$

hold.

Asymptotic bounds and approximation techniques

A simple upper bound

- Plus-selection:
- Population follows the partitions in ascending order
- How long does it take to leave one partition L_i?

Asymptotic bounds and approximation techniques

A simple upper bound

Vacation probability

Let $f : \mathbb{B}^n \rightarrow \mathbb{R}$ be a fitness function and $L_0, \dots, L_k$ be a fitness based partition of f. For a standard bit mutation probability of p and $i \in \{0, 1, \dots, k-1\}$

$$s_i := \min_{x \in L_i} \sum_{j=i+1}^{k} \sum_{y=L_j} p^{H(x,y)} (1-p)^{n-H(x,y)}$$

defines the vacation probability of L_i. In this formula, $H(x, y)$ describes the hamming distance from x to y.

Asymptotic bounds and approximation techniques

A simple upper bound

- Fix x for several y and sum up these probabilities
- Result: probability to mutate from x to one of these y
- Since for $x \in L_i$ summed up y of all L_j with $i < j$:
- Result: probability to leave L_i.
- s_i: Lower bound for the probability to leave L_i with one mutation
- Expected count of mutations until this happens bounded from above by s_i^{-1}.

Asymptotic bounds and approximation techniques

A simple upper bound

A simple Upper bound

Let $f : \mathbb{B}^n \to \mathbb{R}$ be a fitness function and $L_0, \ldots, L_k$ a fitness based partition of f. The expected optimisation time of an $(1+1)$-EA is then bounded from above by

$$E[T_{\mathcal{P}}] \leq \sum_{i=0}^{k-1} s_i^{-1}.$$

Asymptotic bounds and approximation techniques

A simple lower bound

- General bound for evolutionary algorithms
- Requirements:
  - Only mutation as variation operator
  - Standard bit mutation
  - Mutation probability $\frac{1}{n}$
  - Strong unimodal fitness function $f : \mathbb{B}^n \rightarrow \mathbb{R}$

Asymptotic bounds and approximation techniques

A simple lower bound

A simple lower bound

Let $f : \mathbb{B}^n \rightarrow \mathbb{R}$ be a function with exactly one global optimum x^* and A an evolutionary algorithm that initialises its population uniformly at random and utilises only standard bit mutation with mutation probability $p = \frac{1}{n}$. The expected optimisation time of this algorithm is then

$$E[T_{\mathcal{P}}] = \Omega(n \log(n))$$

Asymptotic bounds and approximation techniques

A simple lower bound

Proof.

- Let μ be the population size of A.
- For $\mu = \Omega(n \log(n))$ the algorithm requires already $\Omega(n \log(n))$ evaluations of fitness values for search points prior to finding x^* for the random initialisation of the population with probability $1 - 2^{-\Omega(n)}$.
- When $\mu = O(n \log(n))$, we can see by application of Chernoff bounds that the probability that the hamming distance of a search point x to the optimum x^* is smaller than $\frac{n}{3}$ is

$$P(H(x, x^*) < \frac{n}{3}) = 2^{-\Omega(n)}.$$

Asymptotic bounds and approximation techniques

A simple lower bound

Proof.

- We can therefore assume that at least $\frac{n}{3}$ bits have to be flipped in order to reach the optimum.
- The mutation to flip one bit is $p = \frac{1}{n}$.
- The probability to not flip the bit in t mutations is $(1 - \frac{1}{n})^t \geq e^{-\frac{t}{n-1}}$.
- With $t = (n-1)\ln(n)$ we obtain $e^{-t(n-1)} = \frac{1}{n}$.

Asymptotic bounds and approximation techniques

A simple lower bound

Proof.

- The probability that from $\frac{n}{3}$ bits in t mutations at least one not mutates is therefore at least $1 - (1 - \frac{1}{n})^{\frac{n}{3}} \geq 1 - e^{-\frac{1}{3}}$.
- This leads to

$$E_{T_{\mathcal{P}}} = (1 - 2^{-\Omega(n)}) \cdot (1 - e^{-\frac{1}{3}}) \cdot (m-1)\ln(n) = \Omega(n \log(n)).$$

$\square$

Asymptotic bounds and approximation techniques

The method of the expected progress

- For some problems the optimisation process is similar over whole optimisation run
- Algorithms does not deviate much from expectation in most cases
- Derive lower bound on the optimisation time

Asymptotic bounds and approximation techniques

The method of the expected progress

The method of the expected progress

- Identify steps that are required for the optimisation
- Which are to be applied often on order to reach global optimum
- When we bound the probability to achieve such a step from above, a lower bound can be derived
- With Chernoff bounds bound probability to deviate from the expected number of such steps from above

Asymptotic bounds and approximation techniques

The method of the expected progress

The method of the expected progress

- We denote the optimisation problem with $\mathcal{P}$
- Progress measure: $\mathcal{F} : \mathbb{B}^m \rightarrow \mathbb{R}_0^+$
- $\mathcal{F}(s_t) < \Delta$ means that global optimum not found in first t iterations
- $T_{\mathcal{P}}$: count of iterations required to reach an optimum

Asymptotic bounds and approximation techniques

The method of the expected progress

The method of the expected progress

For every $t \in \mathbb{N}$ we have

$$\begin{aligned} E[T_{\mathcal{P}}] &\geq t \cdot P[T_{\mathcal{P}} > t] \\ &= t \cdot P[\mathcal{F}(s_t) \\ &< \Delta] \\ &= t \cdot (1 - P[\mathcal{F}(s_t) \geq \Delta]) \end{aligned}$$

- With the Markov-inequality: $P[\mathcal{F}(s_t) \geq \Delta] \leq \frac{E[\mathcal{F}(s_t)]}{\Delta}$
- Therefore: $E[T_{\mathcal{P}}] \geq t \cdot \left(1 - \frac{E[\mathcal{F}(s_t)]}{\Delta}\right)$
- Obtain lower bound on the optimisation time by providing expected progress after t iterations